ON USING SAGE TO SOLVE CONSTRAINED OPTIMIZATION PROBLEMS APPLYING THE LAGRANGE MULTIPLIERS METHOD

Maria do Carmo Martins, PhD
CMATI & Department of Mathematics, University of Azores, Portugal
Francisco Martins, PhD
LaSIGE & Department of Informatics, Faculty of Sciences, University of Lisbon, Portugal

Abstract:
This paper combines Calculus and Programming to solve constrained optimization problems common in many areas, notably in Economics. It uses Lagrange multipliers, a well-known technique for maximizing (or minimizing) functions, and the free open-source mathematics software system Sage to compute the maximum (minimum) automatically. Moreover, Sage can be used interactively to work out the solution and to graphically interpret the results, which we find a valuable and practical approach in teaching such techniques to the undergraduate level. In this paper we carry out an exercise describing how these three interdisciplinary areas can work together.

Key Words: Constrained optimization problems; Octave programming language; Lagrange multipliers method

Introduction

Finding the extrema of a function subjected to some restriction is a well-studied problem in Mathematics with multiple, direct applications in Economics. For instance, it is plausible for a company to determine the amount of units of each product it must produce in order to maximizing its profits. Mathematical Analysis and Operational Research come into the rescue by providing tools for solving this kind of problems. Techniques like computing function derivatives or the Dantzig's Simplex algorithm [1] are the recurrent candidates, depending on the problem at hand (i.e., linear vs. non-linear problem). In this paper we stick to the methods provided by Mathematical Analysis, which are usually part of the undergraduate Calculus curricula. The novelty we propose is the using of mathematical software, in particular Sage, to interactively solve the problem and to help in visualizing the solution. We believe that using this approach is more appealing and enlightening for undergraduate students to grasp both the theoretical and the practical aspects of these methods.

Sage [2] is a high-level, open-source programmable system targeted to symbolic computation (amongst other things). Unlike the programming languages Fortran, C, or Java, Sage positions itself at a much higher level of abstraction, providing an environment for finding the exact roots of systems of non-linear equations, compute matrix inverses, integrate and differentiate functions, or plotting solutions' graph of a differential equation, to name a few of its features. We find that mastering such a tool (free of costs) can be of invaluable help, not just for the application we propose in this paper, but for many other subjects (e.g., Algebra, Statistics).

We use a very simple example throughout the paper:

A company wants to find its maximum production level that is modeled by the Cobb-Douglas function [3]

$$f(x,y) = 50x^{3/4}y^{1/4},$$

where x *represents the units of labor and* y *represents the units of capital. Each labor unit costs €150 and each capital unit costs €250. The total expenses for labor and capital cannot exceed €40 000.*

In this paper we recall how to solve this problem using the Lagrange multipliers [4, 5] and then illustrate how Sage becomes very handy in computing and illustrating how the method works.

The main text of the paper is organized as follows. The first section sets the mathematical grounds for the problem and recalls the application of the Lagrange multipliers method; the second section gives a brief introduction to Sage presenting the features needed for solving the running problem; finally, the third section illustrates the usage of Sage to solve, to visualize, and to understand the solution to the problem.

Main Text

Optimizing using Lagrange multipliers method

In general, a problem of finding the local maximum (resp. minimum) of a function $f(x,y)$ subjected to some restriction equation $g(x,y)=c$ can be directly solved by using two systems of equations, one for f and g, another for its first derivatives, provided that both functions have continuous first derivatives. Then, inspecting the second derivative of both functions we can decide whether the critical points correspond to a local maximum or minimum. However, this approach is rather cumbersome.

The Lagrange multipliers method presents a much more elegant solution and can easily be applied in this situation. The Lagrange function Λ (the *Lagrangian*) assumes the general form

$$\Lambda(x,y,\lambda)=f(x,y)+\lambda\big(g(x,y)-c\big), \tag{1}$$

where λ is a new variable, denoted the Lagrange multiplier. Briefly, if $f(x_0,y_0)$ is a maximum of function f, there exists some λ_0 such that the triple (x_0,y_0,λ_0) is a stationary point of the Lagrangian, i.e., a point where the first partial derivatives of Λ are zero. The multiplier λ_0 represents the rate of change in (x_0,y_0) with respect to c. The interested reader may refer to [4, 5] for further details on the Lagrange multipliers method.

For applying the Lagrange multipliers method to our running example, we start by determining the Lagrangian. The constraint in this problem comes from the fact that "*The total expenses for labor and capital cannot exceed €40 000*" that can be captured by the equation $g(x,y)$ defined as follows

$$150x + 250y = 40000$$

Then combining functions f and g as defined in (1), we get the Lagrangian

$$\Lambda(x,y,\lambda) = 50x^{3/4}y^{1/4} - \lambda(150x + 250y - 40000)$$

To solve the problem we need to compute the partial derivatives of Λ, provided below

$$\frac{\partial\Lambda}{\partial x} = -150\lambda + \frac{75y^{\frac{1}{4}}}{2x^{\frac{1}{4}}}$$

$$\frac{\partial\Lambda}{\partial y} = -250\lambda + \frac{25x^{\frac{3}{4}}}{2x^{\frac{3}{4}}}$$

$$\frac{\partial\Lambda}{\partial\lambda} = -150x - 250y + 40000$$

and determine the stationary point of the Lagrangian by solving the system of non-linear equation formed by the partial derivatives (i.e., $\partial\Lambda/\partial x = \partial\Lambda/\partial y = \partial\Lambda/\partial\lambda = 0$) in order to x, y, and λ. Solution to the system is when $x = 200$, $y = 40$, and $\lambda = 0.167185$. The economical interpretation of the Lagrange multiplier is that it represents the percentage of each additional Euro spent on capital that will turn into production. For example, if an additional €10 000 were spent on capital, then it would be translated into $0.167185 \times 10\,000 = 1671.85$ additional units of production. If the problem has more than one constraint, then equation (1) needs to account for the additional restriction function and its multiplier. For a new constraint $h(x,y) = d$ equation (1) would be rewritten as

$$\Lambda(x,y,\lambda,\mu) = f(x,y) + \lambda(g(x,y) - c) + \mu(h(x,y) - d)$$

Sage in a nutshell

Sage convers many areas of Mathematics, namely Algebra, Calculus, Combinatorics, Numerical mathematics, and Number Theory. It is available on Linux, Windows, and Mac OS X operating systems; it is simple to install and present a clean and intuitive environment, making it ideal

for interactive sessions. Its usage can range from a sophisticated calculator to an advanced computer algebra system. Sage is an open source alternative to, and rivals with, Magma [6], Maple [7], Mathematica [8], and MATLAB [9], commercial systems widely used in engineering, science, and economics.

We cover variable and function definitions in order to be able to illustrate some symbolic manipulations, in particular for computing partial derivatives, solving systems of equations, and plotting two and three dimension graphics. Symbolic variables are introduced using the *var* function. For instance, *xVar = var ('x')*, introduces a symbolic name *x* and binds it to variable *xVar*, possibly creating *xVar*. Then, whenever *xVar* is used, it refers to the symbolic name *x*. We usual bind the symbolic name to a variable with the same name. The assignment, *yVar = xVar*, introduces a new variable *yVar* that is bound to the same symbolic name *x*. So, expression *yVar + xVar* evaluates to *2x*.

A function is a binding of a name to an expression. (More advanced functions can be created using the Python language.) The function *tripleSquare* that computes the triple of the square of *x* can be defined as *tripleSquare = 3 * xVar ^ 2*. Expression *tripleSquare (2)* applies function *tripleSquare* to argument *2*, yielding number *12*.

A more interesting feature is plotting the graph of *tripleSquare* between *0* and *5*. The task can be easily achieved using *plot (tripleSquare, (x, 0, 5))*. Many additional arguments may be supplied to function plot, like defining colors, graph title, various captions, but we omit such complexity in this paper.

Function differentiation and system solving finalize out brief tour on Sage. As for the former, we simply ask Sage *tripleSquare.diff(xVar)*. This computes the derivative of *tripleSquare* in order to *xVar* (symbolic name *x*), yielding the expected result *6x*. The latter is a bit more involved; Function solve accepts as its first argument an equation to be solved. For instance, let us find when *tripleSquare* is zero; Expression *solve (tripleSquare == 0, xVar)* will do the trick and solve yields *x = 0*. There are two things to notice: the usage of == (double equal sign) to define an equation; the second argument of solve (*xVar*) indicates the variable in order to which the equation must be solved. In order to solve systems of equations, the first argument is a list of functions. Lists are comma-separated sequences of items (in this case of equations) enclosed in square brackets. The following example is self-explanatory.

solve([x + y == 6, x – y == 4], x, y), yielding *x = 5* and *y = 1*

Applying Sage to solve the problem using the Lagrange multipliers method

Figure 1 shows an excerpt of a Sage interactive session for addressing the running example. First we introduce three symbolic names *x*, *y*, *l* and bind them to three variables of the same name. Thereafter, we functions *f* and *g*. Notice that we define a function for the constraint, not an equation. This way is easier for dealing with *g* later in the session. However, we can rewrite equation *g(x,y) == 40 000* in order to y easily using the solve function. Now we are ready to define the Lagrangian *L*, which is straightforwardly written using *f* and *g*. This line ends with *; L*. It is just a form of asking Sage to print the expression in a more convenient and standard notation. The three partial derivatives are obtained from expression *L*, asking Sage to derive according to some expression variable. In the present case, for instance, *L.diff(y)*, computes $\partial L/\partial y$. Determining the stationary point results from the direct application of the solve function. We provide the three equations involving the partial derivatives and ask for the system to be solved in order to *x*, *y*, and *l*. The expecting result appears clean and easy on the screen. The computation could be further condensed and written in a function that just receives *L* and returns the result. To additionally explore Sage we could plot graphs for the functions in order to better understand and visualize even the mathematical concepts underneath the, sometimes, not well understood notions of partial derivative, system solving, etc.

Lagrange multipliers method

```
x, y, l = var('x,y,l')
```

```
f = 50*x^(3/4)*y^(1/4); f
```

$$50x^{\left(\frac{3}{4}\right)}y^{\left(\frac{1}{4}\right)}$$

```
g = 150*x+250*y; g
```

$$150x+250y$$

```
solve (g==40000, y)
```

$$\left[y=-\frac{3}{5}\,x+160\right]$$

```
L = f - l * (g - 40000); L
```

$$-50(3x+5y-800)l+50x^{\left(\frac{3}{4}\right)}y^{\left(\frac{1}{4}\right)}$$

```
dfdx = L.diff(x); dfdx
```

$$-150l+\frac{75y^{\left(\frac{1}{4}\right)}}{2x^{\left(\frac{1}{4}\right)}}$$

```
dfdy = L.diff(y); dfdy
```

$$-250l+\frac{25x^{\left(\frac{3}{4}\right)}}{2y^{\left(\frac{3}{4}\right)}}$$

```
dfdl = L.diff(l); dfdl
```

$$-150x-250y+40000$$

```
solve([dfdx == 0, dfdy == 0, dfdl == 0], x, y, l)
```

$$\left[\left[x=200, y=40, l=\frac{1}{20}\,5^{\left(\frac{3}{4}\right)}\right]\right]$$

Figure 1: Sage interactive session to solve the running problem

We present in Figure 2 a picture that, we believe, illustrates the solution to the problem. The figure is composed of four graphs: in green there is function *f*; function *g* is depicted in red; in blue there is a plane marking the €40 000 restricting; and, finally, there is a black circle spotting the maximum. Notice that the mark is in the surface of f, and in the plane that is perpendicular to the blue plane and that passes in the intersection between the red and the blue planes. More graphs could be plotted to investigate the solution, but we leave it to the interested reader. For that reason we decided to include (top of Figure 2) the source code for producing and depicted the graph we present, so the reader can further explore the tool.

```
p1=plot3d (f, (x, 0, 400), (y, 0, 50), rgbcolor=(0,1,0))
```

```
p2=plot3d(g, (x,0,400), (y, 0, 50), rgbcolor=(1,0,0))
```

```
z=var('z')
```

```
p3=implicit_plot3d(z==40000, (x,0,400), (y, 0, 50), (z, 0, 40001))
```

```
p4=circle((200,40,f(200, 40)), 15, rgbcolor=(0,0,0), fill=true)
```

```
show(p1+p2+p3+p4)
```

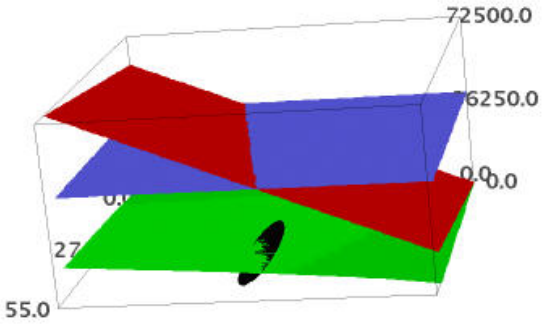

Figure 2: Graph interpreting the solution to the running example

Conclusion

We conducted a small experiment to illustrate the use of a tool, notably Sage, to aid in solving a maximization problem using Lagrange multipliers method. It happens that the problem comes from yet a third area of knowledge and that the mathematical solution can be interpreted using terms from that application domain: Economics. These interdisciplinary quests happen to be important for all the involved areas, since Economics benefits from the tools, both theoretical and practical, that have been researched over the years in Mathematics and Computer Science. On the other hand, Mathematics also benefic from this joint venture, since it gets inspiration in real world problems and then can try to come up with problem drove solutions. Computer Science benefits as well: first there is the opportunity for researching these computed aided systems that tackle all sorts of things automatically; moreover, there is the opportunity to disseminate software by different areas and get actual users performing complex tasks with the tools. Often, and unfortunately, the end users discover most of the bugs, performance issues, etc.

It is our conviction that this kind of tools should be used when teaching undergraduate and graduate courses, since mastering them constitutes a real benefit for expert end users.

References:

[1] Murty, Katta G. *Linear Programming*. John Wiley & Sons, 2002.
[2] Sage web site. http://www.sagemath.org.
[3] Douglas, Paul, The Cobb-Douglas Production Function Once Again: Its History, Its Testing, and Some New Empirical Values. *Journal of Political Economy*, **84**(5):903-916, 1976.
[4] Bertsekas, Dimitri P. Nonlinear Programming (Second ed.). Athena Scientific, 1999.
[5] Vapnyarskii, I. B. Lagrange multipliers, in Hazewinkel, Michiel, Encyclopedia of Mathematics, Springer, 2001.
[6] Magma web site. http://magma.maths.usyd.edu.au/magma/.
[7] Maple web site. http://www.maplesoft.com.
[8] Mathematica web site. http://www.wolfram.com/mathematica/.
[9] MatLab web site. http://www.mathworks.com/products/matlab/.